NAS Kernels on the Connection Machine
RND-90-005

Russell Carter
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA 94035, USA

Revised February, 1991

Introduction

The *NAS Kernel Benchmark Program* is a collection of FORTRAN subroutines, or "kernels", which were chosen to be representative of the computational workload of the Numerical Aerodynamic Simulation (NAS) facility located at NASA Ames Research Center. Details of the kernels are provided in [1]. The kernels provide benchmark measures of double precision (64 bit) floating point computation rate and accuracy and have been run on many different computer systems [2]. The kernels were developed in FORTRAN 77 and were designed for maximum performance on serial vector machines. However, as high performance scientific computing evolves, increasing emphasis has been placed on parallel computer systems. This report details the implementation and performance of the kernels on a massively parallel computer system, a Thinking Machines Corporation Connection Machine 2 (CM) with 32K processors.

Connection Machine Description

The CM system consists of a collection of simple processors, each with its own memory, all acting under the direction of a conventional computer system called the front-end. The Connection Machine system installed at NAS has 32K processors. Each processor has 8K bytes of memory, for a total of 256M bytes of memory. There are 1024 32 bit floating point units. The front-end is a Sun 4/490 with two processors (one dedicated to IO) and 128 megabytes of memory running SunOs 4.0.3. The FORTRAN supplied is CM FORTRAN version 0.7.

The CM performs parallel computations at the FORTRAN program level primarily at the DO loop level. Parallelism in a FORTRAN program is exploited by expressing parallelizable loops in proposed FORTRAN 8x array section syntax. Arrays used in FORTRAN 8x constructions are stored in CM memory, one element per processor. When the front-end computer executes a CM FORTRAN program, it performs serial operations on scalar data stored in its own memory, but sends instructions for array operations to the CM. When the CM receives an instruction, each processor executes it on its own data. The following code fragments illustrate the differences in syntax between the traditional FORTRAN 77 DO loop and the FORTRAN 8x array section syntax.

FORTRAN 77 DO loop version:

```
      do 10 i=1,n
            a(i)=b(i)*c(i)+d(i)*x
            f(i)=cos(a(i))
10    continue
```

FORTRAN 8x version:

```
      a(1:n)=b(1:n)*c(1:n)+d(1:n)*x
      f(1:n)=cos(a(1:n))
```

Procedure

The purpose of the project was to perform a port of the kernels to the CM. Work on the project was initiated by E. Barszcz and L. A. Tanner at NAS. This report extends and completes their work.

No attempt was made to substitute appropriate parallel algorithms for the serial, vectorizable algorithms in the kernels. As few changes to the original FORTRAN source were made as possible. Thus the first step of the port was to convert the parallelizable FORTRAN 77 DO loops to FORTRAN 8x array section syntax. Since the Cray FORTRAN compiler *cft77* supports the proposed array syntax extensions, and the compile-link-run cycle is much faster on the NAS Cray Y-MP than on the CM, the NAS Kernels were first converted to the Cray's version of the array section syntax. Subsequently the converted kernels were run on one processor of the Cray Y-MP 8128, and produced the following results:

Table 1
Cray Y-MP FORTRAN 8x

THE NAS KERNEL BENCHMARK PROGRAM

PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
MXM	1.8085E-13	4.1943E+08	1.5825	265.04
CFFT2D	3.2001E-12	4.9807E+08	11.0693	45.00
CHOLSKY	1.8256E-10	2.2103E+08	4.9587	44.57
BTRIX	6.0622E-12	3.2197E+08	4.4777	71.91
GMTRY	6.5609E-13	2.2650E+08	3.8344	59.07
EMIT	1.5609E-13	2.2604E+08	1.3052	173.18
VPENTA	2.3541E-13	2.5943E+08	7.1444	36.31

TOTAL	1.9305E-10	2.1725E+09	34.3724	63.20

The ERROR column lists the difference between a single computed value and a reference value in each kernel. The FP OPS column lists the number of double precision (64 bit) floating point operations performed in each kernel. The SECONDS column lists the CPU time used to compute each kernel. No initialization work is counted toward CPU time. MFLOPS lists millions of floating point operations per second. The TOTAL row is the sum of each column, except for the MFLOPS entry, which is the total millions of floating point operations divided by the total CPU time used.

Several kernels ran significantly slower when expressed in array section syntax. Comparing output from the unmodified FORTRAN 77 version of the kernels shows decreases in MFLOP computation rate in the CFFT2D, CHOLSKY, BTRIX, GMTRY, and VPENTA kernels. The FORTRAN 77 output of a run on one processor of the Cray Y-MP is provided in Table 2.

Table 2
Cray Y-MP FORTRAN 77

THE NAS KERNEL BENCHMARK PROGRAM

PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
MXM	1.8085E-13	4.1943E+08	1.5705	267.06
CFFT2D	3.2001E-12	4.9807E+08	7.0951	70.20
CHOLSKY	1.8256E-10	2.2103E+08	2.6393	83.75
BTRIX	6.0622E-12	3.2197E+08	2.3717	135.76
GMTRY	6.5609E-13	2.2650E+08	2.0910	108.32
EMIT	1.5609E-13	2.2604E+08	1.2987	174.05
VPENTA	2.3541E-13	2.5943E+08	4.7900	54.16
TOTAL	1.9305E-10	2.1725E+09	21.8563	99.40

Differences in MFLOP rates between the array section syntax version and the original FORTRAN 77 version occur because the *cft77* compiler apparently generates less efficient code for some loops expressed in the array section syntax.

Conversion of Y-MP 8x version to CM FORTRAN

The Y-MP array section syntax version was then converted to Connection Machine FORTRAN. With the exception of two kernels discussed below and the MXM kernel, no modifications were made to the original algorithm. Since general matrix multiplication is a standard CM FORTRAN library call, the CM FORTRAN library matrix multiply routine MATMUL was used to perform the matrix multiplications in the MXM kernel.

Transferring the Y-MP FORTRAN 8x version of the kernels to the CM was straightforward. Most loops expressed in array section syntax ran without additional modification on the CM. Programming issues that had to be addressed in order to permit computations to run on the CM stemmed primarily from the front-end coupled to massively parallel back-end architecture of the CM system. Array initialization in the kernels consists of scalar (sequential) assignment of pseudo-random numbers to the appropriate arrays. Scalar assignments on the CM are performed on the front end; hence all scalar array initializations were performed on the front end and explicitly passed to the CM. This obviated the need for common blocks, so arrays declared in common were redeclared as local arrays within the test subroutines and passed to the kernel subroutines as parameters. Initialization work was not included in the timed sections.

The amount of time spent on this project was about three person-weeks.

Results

The kernels run very slowly on the CM. In order to obtain results in a reasonable amount of real time, the number of iterations for each kernel was reduced to the minimum amount. The number of floating point operations performed decreased accordingly and is reflected in the total floating point operations column (FP OPS) in the output. This modification does not affect either accuracy (as shown in the column labeled ERROR) or MFLOP computation rate. The results of a 32K (all available processors) CM run using double precision (64 bit) floating point arithmetic are provided in Table 3.

Table 3
CM FORTRAN 90

THE NAS KERNEL BENCHMARK PROGRAM

PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
MXM	2.8258E-15	4.1943E+06	4.2946	0.976*
CFFT2D	4.8710E-08	4.9807E+06	75.0830	0.066
CHOLSKY	3.0581E-12	1.1052E+06	29.5347	0.037

BTRIX	4.1941E-13	5.3662E+05	118.9443	0.004
GMTRY	6.9573E-06	1.1325E+08	59.4254	1.905
EMIT	1.0593E-07	2.2604E+07	35.6761	0.633
VPENTA	4.6957E-15	6.4858E+05	24.7714	0.026
TOTAL	7.1119E-06	1.4732E+08	347.7298	0.42

* Performance using library matrix multiplication call MATMUL.

The Y-MP 8x version of the GMTRY, EMIT, and BTRIX kernels ran exceptionally slowly on the CM. Projected elapsed run times obtained from partial runs of the shortened kernels were on the order of several hours. This is the result of the algorithmic characteristics of these kernels. The number of iterations of the parallelizable loops in these kernels range from 5 to 1000. Since the number of processors computing simultaneously is at most the number of the number of parallel iterations of a loop,the maximum computation rate is limited by the number of iterations in the loop. A way of introducing higher level parallelism in these routines is to use the CM FORTRAN library subroutines SPREAD and SUM. These routines, which are not available for the Y-MP, increase the fraction of parallelizable operations by duplicating data across processors. An example from EMIT kernel illustrates this idea. The Y-MP 8x version of the code is given below:

```
      expz(1:nv) = EXP (z(1:nv) * pidp)
      expmz(1:nv) = 1. / expz(1:nv)
      DO 1 l = 1, nb
            DO 2 k = 1, nwall(l)
                  expwkl = EXP (wall(k,l) * pidp)
                  expmwk = 1. / expwkl
                  sps = 0.
                  dum3(1:nv) = expz(1:nv) * expmwk - expwkl * exp-
mz(1:nv)
                  ps(1:nv) = gamma(1:nv) * LOG (REAL(dum3(1:nv)) **
2 +
     &      AIMAG(dum3(1:nv)) ** 2 + sig2)
                  DO 3 i = 1, nv
                        sps = sps + ps(i)
3                 CONTINUE
2           CONTINUE
1     CONTINUE
```

The DO 2 and DO 3 loops are computed sequentially. After converting the code to CM FORTRAN, redeclaring appropriate scalars as arrays, and incorporating the SPREAD and SUM library routines, the code becomes:

```
      expz(1:nv,1) = EXP (z(1:nv) * pidp)
      expmz(1:nv,1) = 1. / expz(1:nv,1)
      expz(1:nv,1:nw)=SPREAD(expz(1:nv,1),2,nw)
      expmz(1:nv,1:nw)=SPREAD(expmz(1:nv,1),2,nw)
      gamma(1:nv,1:nw)=SPREAD(gamma(1:nv,1),2,nw)
      DO 1 l = 1, nb
            sps=0.
            nl=nwall(l)
            expwkl(1,1:nl) = EXP (wall(1:nl,l) * pidp)
            expmwk(1,1:nl) = 1. / expwkl(1,1:nl)
            expwkl(1:nv,1:nl) = SPREAD(expwkl(1,1:nl),1,nv)
            expmwk(1:nv,1:nl) = SPREAD(expmwk(1,1:nl),1,nv)
            dum3(1:nv,1:nl) = expz(1:nv,1:nl) * expmwk(1:nv,1:nl) -
      &     expwkl(1:nv,1:nl) * expmz(1:nv,1:nl)
      &     ps(1:nv,1:nl) = gamma(1:nv,1:nl) *
      &     LOG (REAL (dum3(1:nv,1:nl)) ** 2 +
      &     AIMAG(dum3(1:nv,1:nl)) ** 2 + sig2)
            sps(1:nl)=SUM(ps(1:nv,1:nl),dim=1)
1     CONTINUE
```

The use of SPREAD and SUM allows the parallel computation of the DO 2 and DO 3 loops. These routines were successfully used in GMTRY and EMIT to reduce execution time to the amount reported in Table 3. BTRIX, on the other hand, has a maximum parallel iteration count of 28 and would require much more restructuring to take advantage of these routines. Execution time for the BTRIX routine was reduced by computing only the iteration for which the accuracy check was performed.

Most computational operators, library routines and data assignments appear to work correctly in double precision real and complex formats. However, inferior accuracy of the kernels CFFT2D, GMTRY, and EMIT suggests that complex arithmetic is not performed in double precision.

Two obscure bugs were uncovered during the port. One was a compiler bug that caused compilation to fail inexplicably for a legal CM FORTRAN construct, the other was a runtime error that failed nonlocally, i.e., the program gave evidence of failing at a point in the code that both logically and locationally seemed unrelated to the actual source of error. Both were resolved in impressively short time by the Thinking Machines Corporation representative, and were avoided by minor changes to the source code.

Slowness of the kernels in general can be attributed to the fact that the amount of available parallelism in the kernels is poorly matched to the architecture of the CM, and to the slowness of the software computation of 64 bit floating point. This is best seen by the matrix multiplication kernel MXM. The (presumably optimized) CM FORTRAN library routine MATMUL was used to obtain the following results on 32K processors, and is compared to the FORTRAN 77 implementation of MXM run on single processors of the Cray Y-MP and Cray 2 in Table 4. All values are in MFLOPS.

Table 4

Problem	CM 32bit	CM 64bit	Y-MP 64bit	Cray 2 64bit
A * B				
(256x128)*(128x64)	2.22	0.39	268.66	150.44
(512x256)*(256x128)	14.99	1.17	277.40	172.88
(1024x512)*(512x256)	115.08	3.51	280.45	168.22
(1024x1024)*(1024x512)	454.36	9.55	280.13	158.43
(1024x1024)*(1024x1024)	899.78	17.01	278.78	164.51

The matrix multiplication with dimensions 256x128 and 128x64 is the same size as that of the NAS Kernel MXM matrix multiplication. As the amount of parallelizable work increases, CM MFLOP computation rate increases as well. Codes with relatively small amounts of data such as the NAS kernels can be expected to run poorly on the CM.

Summary

The NAS Kernels were successfully ported to the CM. The programming environment and model is acceptably robust and general. Performance is poor, in part due to the small amount of explicitly parallel work in the standard NAS Kernels, and also because 64 bit computations are computed in software.

Acknowledgment

The author wishes to thank Duane Carbon, Doreen Cheng, John Barton, Robert Bergeron, and David Browning for their constructive comments on this work. The author also wishes to thank Kyra Lowther of Thinking Machines Corporation for her generous and timely assistance and advice with locating and remedying problems I encountered during this work.

References

[1] Bailey, David H., and Barton, John T., "The NAS Kernel Benchmark Program", NASA Technical Memorandum 86711 (August 1985).

[2] Bailey, David H., "NAS Kernel Benchmark Results", *First International Conference on Supercomputing Systems*, IEEE Computer Society, 1985, 341-345.